1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Departamento de Engenharia Informática

Fundamentos de Inteligência Artificial
2024/2025 - 2º Semestre

Trabalho Prático Nº2:
The evolution of Lunar Lander

Nota: A fraude denota uma grave falta de ética e constitui um comportamento inadmissível num estudante do ensino superior e futuro profissional licenciado. Qualquer tentativa de fraude levará à anulação da componente prática tanto do facilitador como do prevaricador, independentemente de ações disciplinares adicionais a que haja lugar nos termos da legislação em vigor. Caso haja recurso a material não original, as **fontes** devem estar explicitamente indicadas. Caso use ferramentas de IA na produção deste trabalho (e.g. ChatGPT), deverá identificar de forma clara todas as partes em que a ferramenta esteve envolvida. Note que, durante a defesa, deverá demonstrar ter conhecimento profundo dos conteúdos gerados pela ferramenta, sendo esse conhecimento objeto de avaliação.

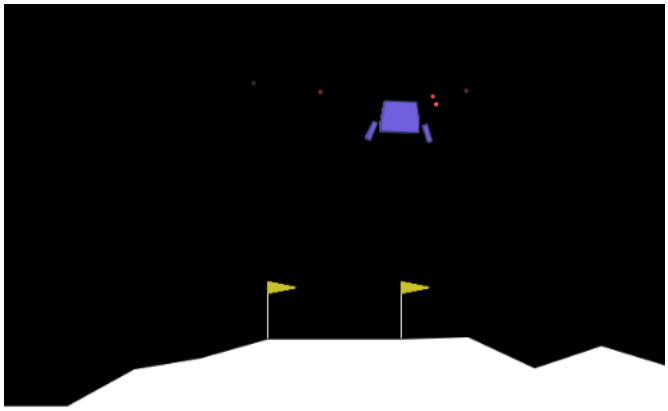

Figura 1: Lunar Lander

1 Introdução

O Lunar Lander é um ambiente de simulação frequentemente utilizado para desenvolver e testar métodos de Inteligência Artificial, tais como aprendizagem por reforço.

No TP1, foi modelado e implementado um agente reativo para aterrar a nave em segurança. Neste trabalho, pretende-se que desenvolva uma abordagem de neuroevolução para o mesmo fim.

2 Lunar Lander

Este trabalho prático tem como objetivo principal a aquisição de competências relacionadas com o desenho e desenvolvimento de Algoritmos Evolucionários para a evolução de agentes autónomos. Para tal, usar-se-á o Lunar Lander como plataforma de aprendizagem. Importa assim descrever quais os componentes fundamentais deste problema, nomeadamente o ambiente e o agente.

2.1 Ambiente

O Lunar Lander consiste num ambiente 2D, contemplando espaço e a superfície lunar, onde está a plataforma de aterragem (nas coordenadas (0,0)). A simulação começa com a nave a pairar sobre a zona central da janela com uma orientação inicial aleatória. O objetivo é desenvolver um agente que aterre a nave de forma segura, controlando os motores.

Para tal, o agente deve lidar com os seguintes desafios:

1. A nave está sujeita à força da gravidade, puxando-a continuamente para baixo.
2. O agente deve manobrar a nave, controlando a sua posição, velocidade e orientação.
3. Para que uma aterragem seja bem sucedida, devem-se verificar as seguintes condições:
   (a) Manter uma velocidade vertical baixa.
   (b) Aterrar com uma orientação aproximadamente vertical.
   (c) Aterrar na zona delimitada pelas duas bandeiras, tendo ambas as "pernas"em contacto com o solo.

A simulação passa-se em tempo discreto, sendo que a cada momento (passo de simulação), o agente recebe uma observação do ambiente, executa uma ação com base no seu controlador e transita para o próximo estado.

2.2 Agente

No presente trabalho, a nave será controlada por uma rede neuronal que tomará como entrada as observações do ambiente e retornará uma ação na forma de comandos dos motores. Assim, importa definir o espaço de observações e ações.

2.2.1 Espaço de Observações

O espaço de observação é representado por um vetor com 8 números reais, codificando o estado da nave no ambiente. A Tabela 1 descreve as componentes deste vetor.

Tabela 1: Espaço de observações.

Índice	Variável	Descrição
0	x	Posição horizontal da nave em relação ao centro (plataforma de aterragem). Negativa à esquerda; positiva à direita.
1	y	Posição vertical da nave em relação ao solo.
2	v_x	Velocidade horizontal da nave. Negativa quando se move para a esquerda; positiva quando se move para a direita.
3	v_y	Velocidade vertical da nave. Negativa quando desce; positiva quando sobe.
4	θ	Orientação da nave. Negativa quando está inclinada para a direita; positiva quando está inclinada para a esquerda.
5	v_θ	Velocidade angular (mudança no ângulo). Negativa quando roda no sentido horário; positiva quando roda no sentido anti-horário.
6	*left_leg_touching*	Booleano (1 ou 0): Indica se a perna esquerda está em contato com o solo.
7	*right_leg_touching*	Booleano (1 ou 0): Indica se a perna direita está em contacto com o solo.

2.2.2 Espaço de Ações

Cada ação é representada por um vetor com dois números reais. O primeiro valor controla o acelerador do motor principal, enquanto o segundo controla os motores secundários.

- **Motor Principal** — É ativado apenas para valores superiores a 0.5, aumentando a sua aceleração de forma linear até atingir o máximo em 1.

- **Motores secundários** — Não são ativados para valores entre -0.5 e 0.5. Para valores inferiores a -0.5, o motor direito é ativado, rodando a nave para a esquerda. Para valores superiores a 0.5, o motor esquerdo é ativado, rodando a nave para a direita. Assim como o motor principal, as suas acelerações aumentam linearmente até os limites de -1 (motor esquerdo) e 1 (motor direito).

Desta forma, é possível controlar os motores de forma progressiva, permitindo ajustes suaves durante o voo ou aterragem.

3 Neuroevolução do Lunar Lander

O objetivo deste trabalho passa por desenvolver as componentes de um Algoritmo Evolucionário para evoluir controladores para o Lunar Lander. Em particular, esses controladores serão redes neuronais com arquiteturas definidas manualmente, sendo apenas os seus parâmetros alvo de evolução.

Para tal, é-lhe fornecido um código de base, que poderá encontrar no *UCStudent*. Este código foi desenvolvido e testado em **Python 3.12**. Recomendamos que utilize a plataforma Anaconda para criar um novo ambiente com esta versão de Python. Pode encontrar instruções de instalação do Anaconda aqui. Posteriormente, deverá instalar a framework Gymnasium com os ambientes Box2D, através do comando:

```
pip install "gymnasium[box2D]"
```

Deverá ainda usar o comando abaixo para instalar o módulo pygame, de forma a poder controlar a nave através do teclado:

```
pip install pygame
```

3.1 Redes Neuronais

No código fornecido, já existe uma implementação de uma rede neuronal simples, como pode ver na listagem abaixo.

```
41 def network(shape, observation,ind):
42     #Computes the output of the neural network given the
           observation and the genotype
43     x = observation[:]
44     for i in range(1,len(shape)):
45         y = np.zeros(shape[i])
46         for j in range(shape[i]):
47             for k in range(len(x)):
48                 y[j] += x[k]*ind[k+j*len(x)]
```

```
49          x = np.tanh(y)
50      return x
```

Esta função recebe como parâmetros a *shape* (ou arquitetura) da rede, a observação atual e o genótipo do indivíduo (*ind*), tomando diretamente como entrada a observação e retornando a ação para os motores.

A arquitetura da rede é definida no cabeçalho do script (conforme a listagem abaixo), sendo representada por um tuplo de valores, sendo o primeiro valor o número de inputs, o último valor o número de outputs, e os valores intermédios os tamanhos das camadas escondidas. No código fornecido, é utilizada apenas uma camada escondida com 12 neurónios.

```
22 nInputs = 8
23 nOutputs = 2
24 SHAPE = (nInputs,12,nOutputs)
25 GENOTYPE_SIZE = 0
26 for i in range(1, len(SHAPE)):
27     GENOTYPE_SIZE += SHAPE[i-1]*SHAPE[i]
```

O tamanho do genótipo é calculado automaticamente a partir da sua arquitectura (i.e., do valor de *SHAPE*), correspondendo ao número de ligações da rede. Note que neste trabalho não são usados *biases*.

3.2 Algoritmo Evolucionário

O ciclo principal de um Algoritmo Evolucionário está já implementado no código fornecido, como pode ver na listagem abaixo. Começa-se pela instanciação de processos para a avaliação concorrente dos indivíduos (linhas 174 a 177), prosseguindo para a criação e avaliação da população inicial (linhas 181 a 184). Segue-se o ciclo evolutivo (linhas 188 a 213) onde, em cada geração, é criado um conjunto de novos indivíduos com recurso a crossover e a mutação (linhas 192 a 202). De seguida, os filhos são avaliados (linha 205) e uma nova população é criada (linha 208).

```
172 def evolution():
173     #Create evaluation processes
174     evaluation_processes = []
175     for i in range(NUM_PROCESSES):
176         evaluation_processes.append(Process(target=evaluate,
                args=(evaluationQueue, evaluatedQueue)))
177         evaluation_processes[-1].start()
178 
179     #Create initial population
180     bests = []
181     population = list(generate_initial_population())
182     population = evaluate_population(population)
183     population.sort(key = lambda x: x['fitness'], reverse=True)
```

```
184     best = (population[0]['genotype']), population[0]['fitness']
185     bests.append(best)
186 
187     #Iterate over generations
188     for gen in range(NUMBER_OF_GENERATIONS):
189         offspring = []
190 
191         #create offspring
192         while len(offspring) < POPULATION_SIZE:
193             if random.random() < PROB_CROSSOVER:
194                 p1 = parent_selection(population)
195                 p2 = parent_selection(population)
196                 ni = crossover(p1, p2)
197 
198             else:
199                 ni = parent_selection(population)
200 
201             ni = mutation(ni)
202             offspring.append(ni)
203 
204         #Evaluate offspring
205         offspring = evaluate_population(offspring)
206 
207         #Apply survival selection
208         population = survival_selection(population, offspring)
209 
210         #Print and save the best of the current generation
211         best = (population[0]['genotype']), population[0]['
                fitness']
212         bests.append(best)
213         print(f'Best of generation {gen}: {best[1]}')
214 
215     #Stop evaluation processes
216     for i in range(NUM_PROCESSES):
217         evaluationQueue.put(None)
218     for p in evaluation_processes:
219         p.join()
220 
221     #Return the list of bests
222     return bests
```

Cada componente do algoritmo evolucionário é implementado na sua própria função. A população inicial é criada pela função mostrada na listagem abaixo. Tal como pode ver, cada indivíduo consiste num dicionário, com um fitness (inicialmente a *None*) e um genótipo. Em particular, o genótipo é uma lista de floats amostrados de uma distribuição uniforme entre -1 e 1, e com comprimento definido por *GENOTYPE_SIZE*.

```
135 def generate_initial_population():
```

```
136     #Generates the initial population
137     population = []
138     for i in range(POPULATION_SIZE):
139         #Each individual is a dictionary with a genotype and a
                fitness value
140         #At this time, the fitness value is None
141         #The genotype is a list of floats sampled from a uniform
                distribution between -1 and 1
142
143         genotype = []
144         for j in range(GENOTYPE_SIZE):
145             genotype += [random.uniform(-1,1)]
146         population.append({'genotype': genotype, 'fitness': None
                })
147     return population
```

Tal como já foi referido, a avaliação é feita de forma concorrente por múltiplos processos. A listagem abaixo mostra a função chamada pelo Algoritmo Evolucionário, que faz uso de duas filas para comunicar com os processos que avaliam cada individuo. Note que é retornada uma nova lista, com os indivíduos devidamente avaliados.

```
125 def evaluate_population(population):
126     #Evaluates a list of individuals using multiple processes
127     for i in range(len(population)):
128         evaluationQueue.put(population[i])
129     new_pop = []
130     for i in range(len(population)):
131         ind = evaluatedQueue.get()
132         new_pop.append(ind)
133     return new_pop
```

Cada processo corre a função *evaluate*, avaliando assim cada indivíduo com base na simulação de um episódio do Lunar Lander. Note que a aptidão de cada indivíduo é calculada pela função *objective_function* (reproduzida na listagem abaixo) utilizando a penúltima observação do ambiente. De modo a assegurar uma funcionalidade básica, foi implementada uma função objectivo simples, que apenas avalia a distância final à base de aterragem. Note que **deverá implementar a sua função objectivo, de forma a evoluir o comportamento desejado**.

```
106 def objective_function(observation):
107     #TODO: Implement your own objective function
108     #Computes the quality of the individual based
109     #on the horizontal distance to the landing pad, the vertical
            velocity and the angle
110     x = observation[0]
111     y = observation[1]
```

```
112     return -abs(x) - abs(y), check_successful_landing(
            observation)
```

Os componentes relativos à seleção de progenitores, crossover e mutação deverão ser implementados por si. No entanto, pelos motivos previamente descritos, foram já criadas funções para cada um deles. A listagem abaixo mostra estas funções.

```
149 def parent_selection(population):
150     #TODO
151     #Select an individual from the population
152     return copy.deepcopy(random.choice(population))
153
154 def crossover(p1, p2):
155     #TODO
156     #Create an offspring from the individuals p1 and p2
157     return p1
158
159 def mutation(p):
160     #TODO
161     #Mutate the individual p
162     return p
```

O mecanismo de seleção de sobreviventes está já implementado, pelo que não terá de lhe fazer alterações. No entanto, é benéfico que o compreenda. Trata-se do mecanismo de seleção elitista, sendo o tamanho da elite definido no cabeçalho do script. A sua implementação é mostrada na listagem abaixo.

```
164 def survival_selection(population, offspring):
165     #reevaluation of the elite
166     offspring.sort(key = lambda x: x['fitness'], reverse=True)
167     p = evaluate_population(population[:ELITE_SIZE])
168     new_population = p + offspring[ELITE_SIZE:]
169     new_population.sort(key = lambda x: x['fitness'], reverse=
            True)
170     return new_population
```

4 Metas

O presente trabalho prático encontra-se dividido em 2 metas distintas. Tal como no trabalho anterior, a primeira meta é de entrega facultativa. Porém, é importante notar que o trabalho relativo à primeira meta tem uma cotação de 90%, sendo a totalidade do trabalho avaliada na entrega final e defesa.

4.1 Meta 1

O objectivo da primeira meta consiste na modelação, implementação e análise de um Algoritmo Evolucionário para a evolução de agentes que aterrem a nave com sucesso em ambientes **sem vento**.

A representação escolhida, os operadores genéticos, os mecanismos de seleção, e a função objectivo, são alguns dos componentes essenciais de um algoritmo genético. Desta forma, a etapa de **modelação** desempenha um papel fundamental no sucesso do seu algoritmo. Relativamente à representação de cada indivíduo, o código fornecido considera que cada genótipo é uma lista de números reais. No entanto, terá ainda de desenvolver alguns dos componentes do AE, escolhendo quais os mecanismos mais apropriados para o problema em causa e implementando-os nos respetivos locais. Em particular, deverá implementar os seguintes componentes:

- Mecanismo de seleção de progenitores
- Operador de crossover
- Operador de mutação
- Função objetivo

Para além destes componentes, deve ainda definir os parâmetros do algoritmo evolucionário que devem ser configurados e alterados no cabeçalho do ficheiro. Os valores de parâmetros tais como a arquitetura da rede, o tamanho da população, número de gerações e probabilidades de crossover/mutação deverão ser escolhidos de forma a atingir resultados satisfatórios em tempo aceitável. A tabela 2 exemplifica as combinações a testar.

Após implementadas, é vital **testar** as funcionalidades do algoritmo evolucionário por forma a garantir o seu bom funcionamento. Nesta meta, deve focar-se na evolução de agentes que aterrem a nave com sucesso na **ausência de vento**, sendo que o sucesso é verificado pela função *check_successful_landing*. **Note que esta parte do trabalho tem uma cotação de 90%**.

Dada a natureza estocástica das abordagens evolucionárias não é possível tirar conclusões a partir de uma única execução. Para cada combinação de parâmetros deverá realizar, pelo menos, **5** repetições da experiência para que a comparação efetuada tenha significado estatístico. A Tabela 2 apresenta as combinações de parâmetros mínimas a testar.

De forma a permitir a análise dos resultados, durante a evolução, são guardados três tipos de informação em ficheiro. Para cada experiência, é criado um ficheiro com o nome *logX.txt*, onde *X* refere-se ao número da experiência. Cada linha destes ficheiros guarda, para cada geração, a aptidão

do melhor indivíduo, a topologia da rede (*SHAPE*) e o seu genótipo. Esta informação não só será útil para analisar o comportamento dos algoritmos, bem como permitirá carregar e utilizar os melhores indivíduos.

Note que não basta enumerar resultados experimentais! Deve fazer uma análise dos mesmos procurando explicar as diferenças encontradas e os comportamentos apresentados.

Na zona do script referente ao main (linha 233 em diante), poderá encontrar o código responsável por evoluir ou validar indivíduos de experiências anteriores. No que toca à evolução, foi já preparado um ciclo que permite efetuar múltiplas experiências de uma só vez, bem como uma lista de *seeds* para o gerador de números pseudo-aleatórios de forma a reduzir a variabilidade das experiências.

<table>
<tr><th></th><th>Mutação</th><th>Crossover</th><th>Elitismo</th><th>Tamanho da População</th><th>Gerações</th></tr>
<tr><td>Experiência 1</td><td>0.008</td><td rowspan="2">0.5</td><td rowspan="4">0</td><td rowspan="8">100</td><td rowspan="8">100</td></tr>
<tr><td>Experiência 2</td><td>0.05</td></tr>
<tr><td>Experiência 3</td><td>0.008</td><td rowspan="2">0.9</td></tr>
<tr><td>Experiência 4</td><td>0.05</td></tr>
<tr><td>Experiência 5</td><td>0.008</td><td rowspan="2">0.5</td><td rowspan="4">1</td></tr>
<tr><td>Experiência 6</td><td>0.05</td></tr>
<tr><td>Experiência 7</td><td>0.008</td><td rowspan="2">0.9</td></tr>
<tr><td>Experiência 8</td><td>0.05</td></tr>
</table>

Tabela 2: Experiências a realizar. Note que, para cada experiência deverá realizar 5 execuções da mesma.

4.2 Meta 2

Nesta meta deve focar-se em evoluir agentes capazes de aterrar a nave com sucesso na presença de vento. Para tal, poderá fazer alterações às várias partes do seu algoritmo, desde a parametrização (incluindo a arquitetura da rede) ao mecanismo de seleção, operadores de variação e função objetivo. A par da implementação, deverá explicitar no relatório o processo bem como as alterações necessárias para evoluir agentes capazes de lidar com condições mais adversas. Deverá também incluir uma análise comparativa com a melhor configuração encontrada na meta anterior.

O trabalho relativo a esta meta tem uma cotação de 10% da nota.

5 Datas e Modo de Entrega

Os grupos têm uma dimensão máxima de 3 alunos. A defesa é obrigatória, bem como a presença de todos os elementos do grupo na mesma.

A entrega da meta 1 é opcional, chama-se no entanto a atenção dos alunos para a importância de concluir atempadamente esta meta. Para efeitos de nota apenas será considerada a entrega final e a defesa.

5.1 Meta 1

Material a entregar:

- Script modificado com as devidas implementações, que devem estar devidamente comentadas.
- Um breve documento (max. 10 páginas), em formato pdf, com a seguinte informação:
  - Identificação dos elementos do grupo (Nomes, Números de Estudante, e-mails, Turma(s) Prática(s))
  - Informação pertinente relativamente a esta meta

Modo de Entrega:

Entrega eletrónica através do Inforestudante.
Data Limite: 17 de Abril de 2025

5.2 Meta 2

Tal como indicado anteriormente, esta entrega será a única que tem um impacto direto na nota. O relatório deve conter informação relativa a **todo** o trabalho realizado. Ou seja, o trabalho realizado no âmbito das metas 1 e 2 deve ser **inteiramente descrito**, por forma a possibilitar a avaliação.

Material a entregar:

- Script modificado com as devidas implementações dos componentes do Algoritmo Evolucionário, devendo o código estar devidamente comentado.

- Um relatório (max. 20 páginas), em formato pdf, com a seguinte informação:
  - Identificação dos elementos do grupo (Nomes, Números de Estudante, e-mails, Turma(s) Prática(s))
  - Informação pertinente relativamente à globalidade do trabalho realizado

Num trabalho desta natureza o relatório assume um papel importante. Deve ter o cuidado de descrever detalhadamente todas as funcionalidades implementadas, dando particular destaque aos problemas e soluções encontradas, justificando as suas escolhas. Deve ser fácil ao leitor compreender o que foi feito e ter por isso capacidade de adaptar/modificar o código.

Conforme pode depreender do enunciado, **experimentação** e **análise** são parte fundamental deste trabalho prático. Assim, deve descrever de forma sucinta mas detalhada as experiências realizadas, os resultados obtidos, analisar os resultados e extrair conclusões.

O relatório deve conter informação relevante tanto da perspetiva do utilizador como do programador. Não deve ultrapassar as 20 páginas, formato A4. Todas as decisões tomadas deverão ser devidamente justificadas e explicadas.

Modo de Entrega:

Entrega eletrónica através do Inforestudante.
Data Limite: 16 de Maio de 2025

6 Bibliografia

1. **Inteligência Artificial: Fundamentos e Aplicações**
   Ernesto Costa, Anabela Simões
2. **Artificial Intelligence: A Modern Approach**
   Stuart Russel, Peter Norvig